BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 982 909 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
01.03.2000 Bulletin 2000/09

(51) Int Cl.[7]: **H04L 29/06**, H04L 12/26

(21) Application number: **99306832.9**

(22) Date of filing: **27.08.1999**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **28.08.1998 GB 9818873**
01.09.1998 GB 9819175
08.10.1998 GB 9821994

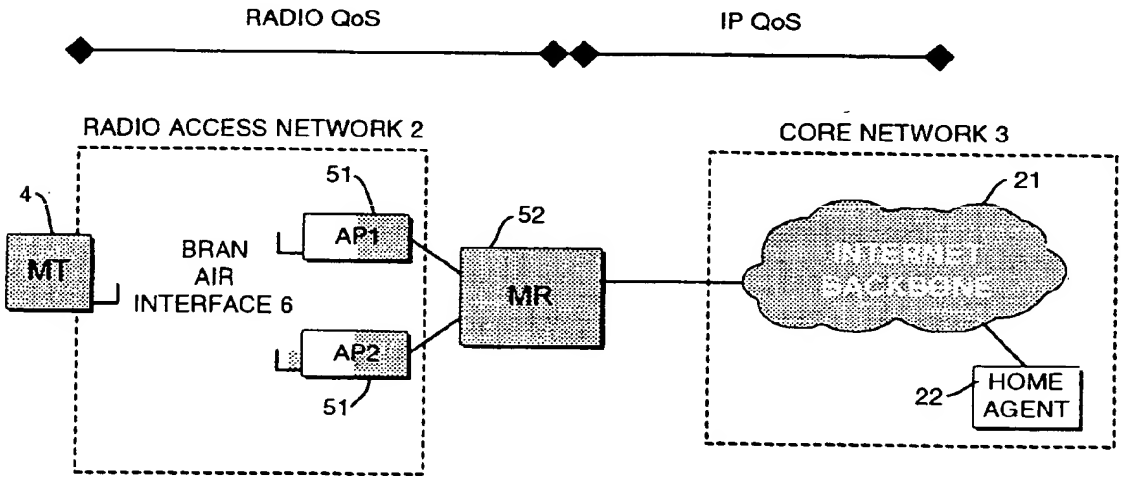
(71) Applicant: **Nokia OYJ**
02150 Espoo (FI)

(72) Inventors:
- **Soderlund, Tom**
  00200 Helsinki (FI)
- **Immonen, Jukka**
  02730 Espoo (FI)

(74) Representative: **Jeffery, Kendra Louise**
Nokia IPR Department,
Nokia House,
Summit Avenue
Farnborough, Hampshire GU14 0NG (GB)

(54) Internet protocol flow detection

(57) An IP flow detector (51; 52) is provided which supports differentiated services in an IP network, such as a wireless IP network. In one embodiment, the detector is arranged to detect a flow type which can be identified by fields in the basic IPv6 header and an extension header. Likewise a method of detecting such IP flows is provided.



MT = MOBILE TERMINAL
AP = ACCESS POINT
MR = MOBILITY ENHANCED IP ROUTER

FIG. 3

EP 0 982 909 A2

Description

[0001] The present invention relates to a mechanism for differentiating between internet protocol (IP) packets, namely IP flow detection. In particular, it relates to a method and apparatus which provide flow detection in packet data transmission.

[0002] Such a mechanism is used, for example, in wireless internet protocol (IP) networks.

[0003] The term "Internet" is commonly used to describe an information resource from which information can be retrieved from a data processor, such as a personal computer (PC). The data processor communicates with the other nodes in the network via e.g., a modem hooked to a telecommunication network. The data processor may be connected to the network also by other means like a direct data network connection. This information resource is worldwide. The Internet is made operable by defining certain data communication standards and protocols, such as TCP (transfer control protocol), UPD (user datagram protocol), and IP (Internet protocol), which are used for controlling data transmission between numerous parts of the Internet. The TCP is involved with providing communicating processes means to transfer a stream of data reliably. The UDP is involved with providing communicating processes means to transfer datagrams reliably. The IP is the common ground for communicating between the nodes in the IP network. The lower part of IP can be modified to suit specific network environments while the upper part of IP as well as the TCP and the UDP above it remain the same everywhere. This way a global uniform network is possible although it has been built on local networks that are based on different technologies. The currently used versions of the Internet protocol are IPv4 and IPv6. IPv4 is defined in RFC791 and IPv6 is defined in the Ipv6 specification dated 8th June 1998.

[0004] Thanks to the growing popularity of open data systems, the Transmission Control Protocol/Internet Protocol (TCP/IP) communication protocol has become a generally used protocol whereby computers of different sizes and brands can communicate with each other. TCP/IP support is currently available for almost any operating system and almost any local network environment. The network layer protocol of TCP/IP, the Internet Protocol IP, is intended to be routed by gateways, i.e. routers. The routing is conducted by means of IP addresses and routing tables. Thanks to the TCP/IP protocol suite, the applications running in the hosts connected to the Internet are able to communicate even though the hosts were located in different continents or even space.

[0005] The rapid evolution of the Internet services has created a strong need for broadband networks with high data rate and Quality of Service (QoS). Video broadcasting and other multimedia distribution services are evolving rapidly. The users are willing to access these services also in the wireless environment. Currently, in the fixed IP network, IP packets are typically sent as best effort data traffic. In the case of network congestion, all data streams are handled with equal priority which may have a dramatic effect on multimedia services. Two main problems exist: firstly the current wireless networks do not provide sufficient QoS mechanism, and secondly, the existing wireless networks are not capable of serving several simultaneous connections with high data rate and QoS requirements. To meet the increased customer requirements, new wireless broadband network techniques are required.

[0006] The Internet Engineering Task Force (IETF) is an organisation involved with the development of the architecture, protocols and operation of the Internet. IEFT has defined two different QoS concepts: integrated and differentiated services, for providing a standard mechanism for supporting real time applications in IP networks. Integrated services is based on an abstract flow model with reservation protocol (RSVP, RFC2205) and admission control. The network reserves statically resources for real time connections in each network device, and hence is not optimally efficient. Consequently the differentiated services concept was developed. This concept is based on the use of an IP header for indicating the requested service class (called per-hop behaviour) for the packet. As a result, each IP packet carries QoS information and no reservations are required. Whilst the IETF suggest the use of an IP header for indicating the QoS, the actual packet handling mechanisms will not be standardised.

[0007] One mechanism used for differentiating between IP packets is IP flow detection. The flow detection concept comprises the monitoring of IP traffic to be able to detect packets flowing frequently between two communicating processes (IP applications). Such packets establish an IP flow.

[0008] The flow detection entity (called the flow classifier) monitors the IP packets and some specific IP header fields in detecting the flows. There are several header fields (parameters) which can be used in flow detection. Figures 1(a) to (c) present the IPv6, TCP and UDP protocol headers respectively and the header fields applicable to flow detection.

[0009] As can be seen from Figure 1(a), the IPv6 header block consists of the following elements:

Version	IP version of 4 bits (=6)
Traffic Class	8 bit priority,
Flow label	20 bit label for identifying the connection in the application layer,
Payload length	16 bit integer indicating the length of the payload, *i.e.* the length of the packet after the header in bytes,
Next header	data of 8 bits determining the header immediately following the IPv6 header,

(continued)

Hop limit	integer counter of 8 bits which is reduced by one at the each device (node) which transmits the packet further; the packet is rejected if the value is reduced to zero,
Source address	the 128 bit address of the sender of the original packet,
Destination address	the 128 bit address of the intended recipient.

[0010] The header is followed by the payload block, *i.e.* the actual information to be transmitted.

[0011] In this IP version, if a packet is provided with a non-zero IPv6 flow label, then the flow label in the IPv6 header together with the source address is a flow identifier (first flow identification type), and directly distinguishes the different IP sessions. The flow label is used by the applications to "mark" the packets belonging to their IP flow. However, this flow label is not available in all IP headers (e.g. IPv4). Moreover, if a system does not support it, the flow label is set to zero. In these instances, fields of an upper layer protocol header can be used as flow identifiers instead, to distinguish different IP sessions. For example, as is shown in Figure 1b, an alternative flow ID can include TCP/UDP source and destination port information to distinguish different IP sessions together with source address and destination address from the IP header (second flow identification type).

[0012] The present inventors have realised the need for a method of flow detection which provides an improved flexibility (for example so its use is not solely dependent upon the use of current protocols).

[0013] According to one aspect of the present invention there is provided a method for detecting an IP flow in flow label deprived packet data transmission, comprising monitoring a set of fields in a lower layer header of the packets to detect an IP flow, wherein monitoring the set of fields comprises monitoring an source address field; monitoring a destination address field; and monitoring a further field indicative of packet management criteria. A lower layer header is generally one from OSI layers 1 to 3 (physical, data link and network layers) and may for example be an IP header or an IP extension header. This IP flow detection method supports the aforementioned QoS concepts for example, when an upper layer header is not available. Consequently, it provides effective flow management.

[0014] According to another aspect of the present invention, there is provided a method of detecting an IP flow in packet data transmission, comprising selecting a set of fields to be monitored to detect an IP flow; and monitoring the selected set of fields in a header of the packets to detect an IP flow; wherein the set of fields is selected from a first set comprising a flow label field and a source address field from a lower layer header of the packets; a second set comprising a source address field and a destination address field from a lower layer header of the packets and a source port field and a destination port field from an upper layer header of the packets; and a third set comprising the source address field, destination address field, and a further field indicative of packet management criteria (other than the flow label field) from the lower layer header of the packets. This IP flow method uses different options, depending on what headers are available. Again, these options provide a flexible method of providing effective flow management.

[0015] Preferably the set of fields are selected in the aforementioned order, as they are ranked having consideration to the complexity and load in routing devices. That is, the lower the priority the increased complexity and load in the routing devices.

[0016] The method may be used for detecting an IP flow encrypted packet data transmission. This is one example when not all the headers are accessible, and in which the present invention is particularly useful. In such a case, the step of monitoring the further field comprises monitoring a security field, such as a security parameter index of the encapsulated security payload header.

[0017] According to a further aspect of the present invention, there is provided the use of a set of fields in a lower layer header of data packets as a flow identifier, wherein the set of fields comprises source and destination address fields and a field indicative of packet management criteria (other than a flow label field).

[0018] According to yet another aspect of the present invention, there is provided an IP flow detector which implements any of the methods above.

[0019] Embodiments of the present invention will now be described, by way of example, of which:

Figure 1a illustrates an IPv6 header format;
Figure 1b illustrates a TCP header format;
Figure 1c illustrates a UDP header format;
Figure 2 illustrates an IPv6 header format when encapsulated security payload (ESP) is used; and
Figure 3 illustrates a general radio system architecture in which the method and apparatus of the present invention may be implemented.

[0020] In the present invention, flow detection is made more flexible by the provision of an alternative flow identifier to the first and second types mentioned above, which caters for the situation where a flow label and/or port addresses

are not available, and yet still supports differentiated services. A flow label may not be available for the following reasons, for example. Firstly, the system may be using a protocol which does not define a flow label field (e.g. IPv4). Secondly, the system may not support flow labels, despite using a protocol which defines a flow label field (e.g. IPv6). In this latter case, the flow label value is set to zero.

[0021] Further, an example of a situation when neither a flow label nor port addresses are available, may be when the data packet is encrypted for security reasons. Encryption of IP packets is discussed in the draft on IP Encapsulating Security Payload (ESP) by Stephen Kent and Randall Atkinson, dated March 1998. Figure 2 of the accompanying drawings illustrates an ESP extension header.

[0022] IP encryption utilises Encapsulating Security Payload (ESP), ESP provides means for encrypting the contents of an IP packet. When ESP based encryption is used (transport-mode ESP), the processing nodes can interpret only the basic IP header and the extension headers preceding the ESP header. This is because transport-mode encapsulates a transport-layer (e.g., UDP, TCP or ICMP) frame inside the ESP. In transport-mode ESP, the ESP header follows the end-to-end headers (e.g., Authentication Header) and immediately precedes an upper layer (e.g., UDP, TCP, ICMPv6) header.

[0023] The encrypted data includes both the protected ESP header fields and the protected user data, which is either an entire IP packet or an upper-layer protocol frame (e.g., TCP or UDP). Thus, the second flow identification type cannot be used because there is no port information available. If, in addition, the flow label is not set, effective flow management cannot be achieved.

[0024] In a preferred embodiment of the present invention, an alternative flow identifier is used which makes use of a source address field, destination address field of the basic IPv6 header and also a further field which is indicative of packet management criteria, and thus provide effective flow detection. On their own, the source and destination addresses do not provide any differentiation and thus do not provide efficient and effective flow management. (Hosts may have several applications running, each application requiring different treatment in the network). In contrast, in the present example of the invention, the Security Parameter Index (SPI) value from the ESP header is used for flow identification. This header is indicative of packet management criteria.

[0025] The SPI value together with the source and destination addresses can be used to identify different IP flows. The SPI field is a 32-bit arbitrary value which together with the destination IP address and security protocol (here ESP) identifies the Security Association for the packet. Figure 2 shows the IP packet format when ESP encryption is used (the IPv6 header fields applicable to flow detection are highlighted in Figure 1).

[0026] When IP encryption is applied (transport-mode ESP), only the information contained in the basic IPv6 header and the extension headers preceding the ESP header can be used for flow detection. From the basic header, flow label, source address and destination address parameters are usable. If flow label values are not used, flow identification granularity can be improved by using the SPI value from the ESP header, together with the source and destination IP addresses to identify a flow. That is, the information available to differentiate IP flows from each other can be increased.

[0027] Thus the solution for detecting flows also for secured IP traffic to provide a third flow identification type as follows:

- type 3: source address + destination address +[SPI]

[0028] Instead of using only the source and destination IP addresses to identify the flow the packet belongs to, it is proposed to also use the SPI value, if there is one available in the extension headers. The flow identification granularity depends on the granularity of the respective Security Association that is identified by the SPI value, the destination IP address and the security protocol (ESP). For example, there may exist several Security Associations between two hosts or only one Security Association. If there was only one Security Association between the two hosts all the different TCP flows would be detected by the proposed algorithm as only one flow because the identifying information in the packets of these flows would be the same (same IP addresses and same SPI value in the ESP header). If however there are several Security Associations between these two hosts e.g., per TCP port basis, the proposed flow detection algorithm would detect each TCP flow as a separate flow because the packets of each flow would have different identifying information (same IP addresses, different SPI values in the ESP header).

[0029] The function that handles the flow detection is called the flow classifier. The flow classier decides when subsequent IP packets with the same flow identifying information should be considered as a flow. Usually some measurements are made at this point to ensure that the packets that belong to the detected flow will be handled specially. For example, in the case of a wireless media, the flow could be bound to a radio connection.

[0030] The flow classifier decides also when the flow should be terminated. The flow will be terminated when no packets with the same flow identifying information are detected within a reasonable time. At this point the resources will be freed that were reserved for the flow when it was detected. For example, in the case of a wireless media, the radio connection would be freed that was reserved for the packets that belonged to the terminated flow.

[0031] Flow classification works so that the IP traffic and specific header fields are monitored in order to detect new flows. In a preferred embodiment of the present invention, the classifier specifies a number of different flow types which

can be used, depending on the IP and transport protocol header fields. For example, the following four different flow types can be specified:

1. Flows identified by flow labels (type 1)
2. Flows identified by TCP/UDP port numbers (type 2)
3. Flows identified by the source and destination IP addresses + the security parameter index (type 3)
4. Flows identified by the source and destination IP addresses (type 4)

[0032] The first option can be applied if the applications are able to use the IPv6 flow label to mark the different IP sessions. If such advanced applications are not available, and if TCP/UDP port information is available, flow type 2 is selected. In case IP encryption is used, the second option cannot be applied since the port information is encrypted. In such a case, security parameter index (SPI) is used with source and destination addresses to identify possible flows. If no TCP/UDP port information, flow labels or SPI parameters are available, the only option is to look just for the source and destination IP addresses and separate flows between hosts (the first two options separate flows in the granularity of communicating processes).

[0033] Each flow type specifies the set of fields from the IP packet header that are used to identify a flow. The set of the header fields identifying a particular flow is called the flow identifier. Depending on the flow type, the flow identifiers contain the following fields:

- Type 1: source address + flow label
- Type 2: source address + destination address + protocol (next header) + source port + destination port
- Type 3: source address + destination address + security parameter index (SPI)
- Type 4: source address + destination address

[0034] In this embodiment, the flow types have been prioritised in the above order to minimise the required load and processing. The first case can be applied when the packets have a nonzero IPv6 flow label, distinguishing directly the different IP sessions. This uses the least processing. As mentioned above, if the flow label is not available but TCP/UDP port information is available instead, the second case is selected. However, this second option requires UDP/TCP header processing allowing efficient flow management but at the same time increasing the complexity and load in routing devices. If neither flow label nor TCP/UDP port information is available, the flows can be identified by the source and destination addresses and the SPI. Since this SPI value is in an extension header, it again requires an increased load over type 1.

[0035] A flow detector can differentiate between these four flow types, and based on the flow classification mechanism bind each flow type to a flow. Three different flow classifier mechanisms which may be applied in the present system are:

- X/Y classifier, meaning X packets (with the same flow identifier) in Y seconds resulting in a new flow
- Protocol classifier which simply assigns all TCP packets to flows
- Port classifier, using transport layer port numbers to decide which flows to bind.

[0036] The X/Y classifier is the preferred choice as it is the only one which supports flow types 1 and 2.

[0037] Typical flow detection criteria for the X/Y classifier are listed in table 1 below. The table gives values for X and Y in a function of different amount of flow space available (e.g. the flow space may refer to the amount of radio connections required in a wireless internet system). Expected performance means the portion of packets switched to flows.

[0038] As can be seen, the values are somewhat different in different environments. Therefore, it should be possible to change easily the values of X and Y in the WFMP implementation.

Table 1: X/Y classifier recommendations

Flow space req.	Gateway	Campus/Enterprise Backbone
1K	Classifier: X = 5 / Y = 15 sec. Flow deletion delay: 30-120 sec. Expected performance: 85%	Classifier: X = 40 / Y = 40 sec. Flow deletion delay: 30-60 sec. Expected performance: 79%
2K	Classifier: X = 5 / Y = 60 sec. Flow deletion delay: 30-120 sec. Expected performance: 90%	Classifier: X = 10 / Y = 45 sec. Flow deletion delay: 30-60 sec. Expected performance: 89%
8K	Classifier: X = 2 / Y = 60 sec. Flow deletion delay: 30-120 sec. Expected performance: 93%	Classifier: X = 5 / Y = 60 sec. Flow deletion delay: 30-60 sec. Expected performance: 92%
16K	Classifier: X = 2 / Y = 60 sec. Flow deletion delay: 30-120 sec. Expected performance: 93%	Classifier: X = 2 / Y = 60 sec. Flow deletion delay: 30-60 sec. Expected performance: 95%
32K	Classifier: X = 2 / Y = 60 sec. Flow deletion delay: 30-120 sec. Expected performance: 93%	Classifier: X = 2 / Y = 60 sec. Flow deletion delay: 30-60 sec. Expected performance: 95%
∞	Classifier: all packets Flow deletion delay: ∞ Expected performance: 99%	Classifier: all packets Flow deletion delay: ∞ Expected performance: 98%

[0039] Since the establishment of a TCP connection always contains at least three packets used, and since the flow detection should be based on actual data packets, a minimum value of six for X is considered appropriate (third data packet triggering the flow detection). The value for Y could be 30 seconds.

[0040] A flow is deleted after some constant number of seconds of inactivity. When flow classifier detects a new flow, it starts the flow inactivity timer. This timer is re-started each time a packet belonging to that flow is received. Once the timer expires, the flow identifier is removed from the list of monitored packets. Finally, the IP flow is released.

[0041] One implementation of the internet protocol is in wireless networks. One such network is shown in Figure 3 of the accompanying drawings. The broadband radio access network 1 (BRAN) is composed of a radio access network 2 having mobile terminals 4, access points 51, 51' and an air interface between, plus a mobility enhanced IP router 52 (M-Router). The BRAN is connected to the core IP network which comprises the internet backbone 21 and home agents 22.

[0042] The radio access network 2 (RAN) implements all the radio dependent functionality such as radio resource management, setup and release of wireless flows, handovers and packet compression. It contains mobile terminals and access points. The mobile terminal 4 is the user's communication device for accessing wireless Internet services,

and is the end point of the Internet and radio access network control protocols. The access point 51,51' implements all the radio dependent control functionality, such as radio resource management. It includes radio resource management and radio link control functions. The corresponding network elements in GSM are the base transceiver stations (BTS/TRX) and base station controller (BSC).

[0043] The M-Router 52 creates the wireless IP sub-network managing one or more access points. The M-Router handles the mobility and location management of the terminals that are registered to the access points 51,51'. The M-Router provides IP mobility services, such as DHCP (dynamic host configuration protocol). DHCP is used for allocating IP addresses for the terminals. The corresponding element to the M-router in the GSM network is the gateway GPRS support node (GGSN). The access points 51, 51' and the terminals 4 with an IP stack that belong to the same IP sub-network (use the same M-ROUTER) create a logical link.

[0044] The core network 3 comprises a home agent 22 which resides in the home network of an associated terminal 4 and is accessed through standard IP gateways. Typically home agent 22 is implemented as part of the M-Router 52 of the home network. However, it can also be a separate entity (e.g. PC host). The home agent 22 can contain user authentication information and a billing database. It resembles the home location register (HLR) in GSM.

[0045] In a preferred embodiment the M-router 52 provides IP flow classification. The network can assign certain quality of service characteristics for a flow, which are required for multimedia service implementations in IP networks. For instance, a particular flow can be prioritised in the router. In the present embodiment, the M-router maintains IP flow QoS characteristics in the air interface and permits the prioritisation of different IP packet (flows) in the radio link. It does this by mapping the detected IP flows into corresponding radio flows for transmission over the RAN. These radio flows have corresponding identifiers and QoS characteristics, and are further discussed in Finnish patent application number 980191 published on 28th July 1999.

[0046] Alternatively, the IP flow classification could be positioned into the access point controller or even into each access point.

[0047] Moreover the criteria used for flow classification in the preferred embodiment to detect an IP flow is not essential to the invention. If IP flow detection is required, then various other criteria can be used. For example, the flow classifier can be dynamically configured by changing the value of the packets/sec detection criteria parameter.

[0048] The present invention includes any novel feature or combination of features disclosed herein either explicitly or any generalisation thereof irrespective of whether or not it relates to the claimed invention or mitigates any or all of the problems addressed.

[0049] In view of the foregoing description it would be evident to a person skilled in the art that various modifications may be made within the scope of the invention. For example, QoS characteristics (such as priority/real time/accuracy) were given as the packet management criteria in the preferred embodiment. However, packet management criteria can be based on other factors for which the network requires differentiation.

Claims

1. A method for detecting an IP flow in flow label deprived packet data transmission, comprising monitoring a set of fields in a lower layer header of the packets to detect an IP flow, wherein monitoring the set of fields comprises:

   monitoring an source address field;
   monitoring a destination address field; and
   monitoring a further field indicative of packet management criteria.

2. A method of detecting an IP flow in packet data transmission, comprising:

   selecting a set of fields to be monitored to detect an IP flow; and
   monitoring the selected set of fields in a header of the packets to detect an IP flow;

   wherein the set of fields is selected from:

   a first set comprising a flow label field and a source address field from a lower layer header of the packets;
   a second set comprising a source address field and a destination address field from a lower layer header of the packets and a source port field and a destination port field from an upper layer header of the packets; and
   a third set comprising the source address field, destination address field, and a further field indicative of packet management criteria other than the flow label field from the lower layer header of the packets.

3. A method as claimed in claim 2, wherein the set of fields are selected on the basis of priority, in which the first set

has the highest priority, followed by the second set, and then the third set.

4. A method as claimed in claim 2 or 3, wherein selection of the set of fields to be monitored is determined by the availability of the fields.

5. A method as claimed in any of claims 2 to 4, wherein monitoring the third set of fields comprises:

monitoring an source address field;
monitoring a destination address field; and
monitoring a further field indicative of packet management criteria other than the flow label field.

6. A method as claimed in claim 1 or 5 for detecting an IP flow in encrypted packet data transmission, wherein monitoring the further field comprises monitoring a security field.

7. A method as claimed in claim 6, wherein the encryption utilises encapsulating security payload, and monitoring the further field comprises monitoring a security field in an encapsulating security payload header.

8. A method as claimed in claim 7, wherein monitoring the further field comprises monitoring a security parameter index of the encapsulating security payload header.

9. A method as claimed in any preceding claim, wherein monitoring the set of fields comprises monitoring the set of fields in the basic and extension headers of the packets.

10. Use of a set of fields in a lower layer header of data packets as a flow identifier, wherein the set of fields comprises source and destination address fields and a field indicative of packet management criteria other than a flow label field.

11. An IP flow detector for detecting an IP flow in flow label deprived packet data transmission, comprising a monitor for monitoring a set of fields in a lower layer header of the packets to detect an IP flow, wherein the monitoring means comprises:

a monitor for monitoring an source address field;
a monitor for monitoring a destination address field; and
a monitor for monitoring a further field indicative of packet management criteria.

12. An IP flow detector for detecting an IP flow in packet data transmission, comprising:

a selector for selecting a set of fields to be monitored to detect an IP flow; and
a monitor for monitoring the selected set of fields in a header of the packets to detect an IP flow;

wherein the selector selects the set of fields from:

a first set comprising a flow label field and a source address field from a lower layer header of the packets;
a second set comprising a source address field and a destination address field from a lower layer header of the packets and a source port field and a destination port field from an upper layer header of the packets; and
a third set comprising the source address field, destination address field, and a further field indicative of packet management criteria from the lower layer header of the packets other than the flow label field

13. A detector as claimed in claim 12, wherein selector selects the set of fields on the basis of priority, in which the first set has the highest priority, followed by the second set, and then the third set.

14. A detector as claimed in claim 12 or 13, wherein the selector determines the set of fields to be monitored on the availability of the fields.

15. A method as claimed in any of claims 12 to 14, wherein the monitor for monitoring the third set of fields comprises:

a monitor for monitoring an source address field;
a monitor for monitoring a destination address field; and

a monitor for monitoring a further field indicative of packet management criteria other than the flow label field.

16. A detector as claimed in claim 11 or 16 for detecting an IP flow in encrypted packet data transmission, wherein the monitor for monitoring the further field is arranged to monitor a security field.

17. A detector as claimed in claim 2, wherein the encryption utilises encapsulating security payload, and the monitor for monitoring the further field is arranged to monitor a security field in an encapsulating security payload header.

18. A detector as claimed in claim 3, wherein the monitor for monitoring the further field is arranged to monitor a security parameter index of the encapsulating security payload header.

19. A detector as claimed in any preceding claim, wherein the monitor for monitoring the set of fields in a lower layer header of the packets to detect an IP flow is arranged to monitor the set of fields in the basic and extension headers of the packets.

VERSION	TRAFFIC CLASS	FLOW LABEL	
PAYLOAD LENGTH		NEXT HEADER	HOP LIMIT
SOURCE ADDRESS			
DESTINATION ADDRESS			

FIG. 1(a)

SOURCE PORT			DESTINATION PORT
SEQUENCE NUMBER			
ACKNOWLEDGEMENT NUMBER			
DATA OFFSET	RESERVED	CONTROL BITS	WINDOW
CHECKSUM			URGENT POINTER
OPTIONS			PADDING

FIG. 1(b)

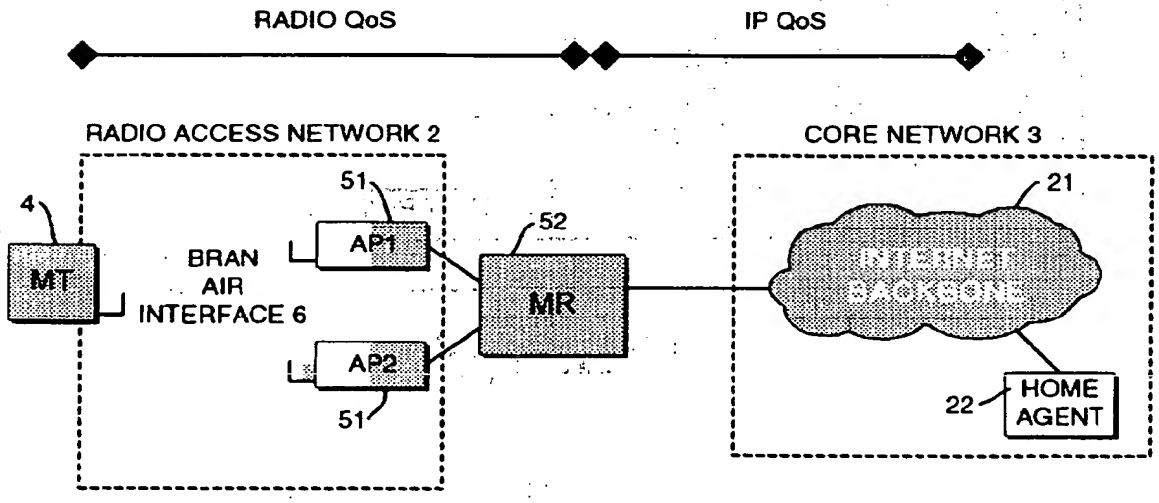
SOURCE PORT	DESTINATION PORT
LENGTH	CHECKSUM

FIG. 1(c)

FIG. 1, IPv6, TCP AND UDP HEADER FORMATS

BASIC IPv6 HEADER
HOP-BY-HOP OPTIONS HEADER
SECURITY PARAMETER INDEX (SPI)
IP PAYLOAD (ENCRYPTED)

FIG. 2 IPv6 HEADER FORMAT WHEN ESP USED



MT = MOBILE TERMINAL
AP = ACCESS POINT
MR = MOBILITY ENHANCED IP ROUTER

FIG. 3

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 982 909 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
29.05.2002 Bulletin 2002/22

(43) Date of publication A2:
01.03.2000 Bulletin 2000/09

(21) Application number: **99306832.9**

(22) Date of filing: **27.08.1999**

(51) Int Cl.[7]: **H04L 29/06**, H04L 12/26

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **28.08.1998 GB 9818873**
01.09.1998 GB 9819175
08.10.1998 GB 9821994

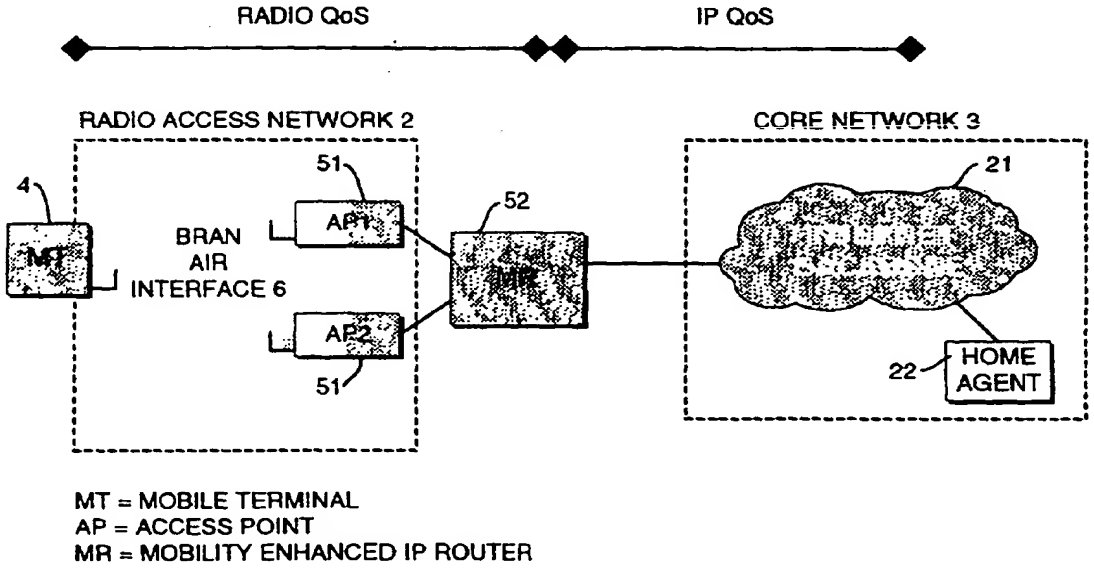
(71) Applicant: **Nokia OYJ**
02150 Espoo (FI)

(72) Inventors:
- **Soderlund, Tom**
  00200 Helsinki (FI)
- **Immonen, Jukka**
  02730 Espoo (FI)

(74) Representative: **Jones, Kendra Louise**
Nokia IPR Department,
Nokia House,
Summit Avenue
Farnborough, Hampshire GU14 0NG (GB)

(54) Internet protocol flow detection

(57) An IP flow detector (51; 52) is provided which supports differentiated services in an IP network, such as a wireless IP network. In one embodiment, the detector is arranged to detect a flow type which can be identified by fields in the basic IPv6 header and an extension header. Likewise a method of detecting such IP flows is provided.



MT = MOBILE TERMINAL
AP = ACCESS POINT
MR = MOBILITY ENHANCED IP ROUTER

FIG. 3

EP 0 982 909 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 6832

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X	NEWMAN P ET AL: "Flow labelled IP: a connectionless approach to ATM" PROCEEDINGS OF IEEE INFOCOM 1996. CONFERENCE ON COMPUTER COMMUNICATIONS. FIFTEENTH ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. NETWORKING THE NEXT GENERATION. SAN FRANCISCO, MAR. 24 - 28, 1996, PROCEEDINGS OF INFOCOM, L, vol. 2 CONF. 15, 24 March 1996 (1996-03-24), pages 1251-1260, XP010158197 ISBN: 0-8186-7293-5	1,10,11	H04L29/06 H04L12/26
Y	* page 1254, right-hand column, line 1 - line 5 * * page 1256, paragraph 5.2. * * page 1257, paragraph 5.6. * * page 1258, line 18 - line 28 * * page 1259, left-hand column, line 19 - right-hand column, line 6 * * page 1259; table 1 *	2-9, 12-19	
Y	IETF: "IPv6 Specification" RFC 1883, 'Online! December 1995 (1995-12), XP002189935 Retrieved from the Internet: <URL:http://ietf.org/rfc/rfc1633.txt?number=1883> 'retrieved on 2002-02-11! * page 29, line 1 - line 17 *	2-5, 12-15	TECHNICAL FIELDS SEARCHED (Int.Cl.7) H04L
Y	IETF: "RSVP Extensions for IPSEC Data Flows" RFC 2207, 'Online! September 1997 (1997-09), XP002189936 Retrieved from the Internet: <URL:http://ietf.org/rfc/rfc1633.txt?number=2207> 'retrieved on 2002-02-11! * page 3, paragraph 2. *	6-9, 16-19	

-/--

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
MUNICH	13 February 2002	Lebas, Y

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 6832

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
A	MARTIGNONI S ET AL: "Extension of classical IP over ATM to support QoS of the application level" 1998 1ST. IEEE INTERNATIONAL CONFERENCE ON ATM. ICATM'98. CONFERENCE PROCEEDINGS. COLMAR, FRANCE, JUNE 22 - 24, 1998, IEEE INTERNATIONAL CONFERENCE ON ATM, NEW YORK, NY: IEEE, US, 22 June 1998 (1998-06-22), pages 492-499, XP010290983 ISBN: 0-7803-4982-2 * page 492, paragraph 1. * * page 494 * * page 496, right-hand column, line 16 - page 497, right-hand column, line 8 *	1-19	
A	IETF: "Integrated Services Architecture" RFC 1633, 'Online! June 1994 (1994-06), XP002189937 Retrieved from the Internet: <URL:http://ietf.org/rfc/rfc1633.txt?number=1633> 'retrieved on 2002-02-11! * paragraph '4.1.3.! *	1-19	TECHNICAL FIELDS SEARCHED (Int.Cl.7)
A	HAO CHE ET AL: "Adaptive resource management for flow-based IP/ATM hybrid switching systems" INFOCOM '98. SEVENTEENTH ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. PROCEEDINGS. IEEE SAN FRANCISCO, CA, USA 29 MARCH-2 APRIL 1998, NEW YORK, NY, USA,IEEE, US, 29 March 1998 (1998-03-29), pages 381-389, XP010270300 ISBN: 0-7803-4383-2 * paragraph '002.! *	1-19	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
MUNICH	13 February 2002	Lebas, Y

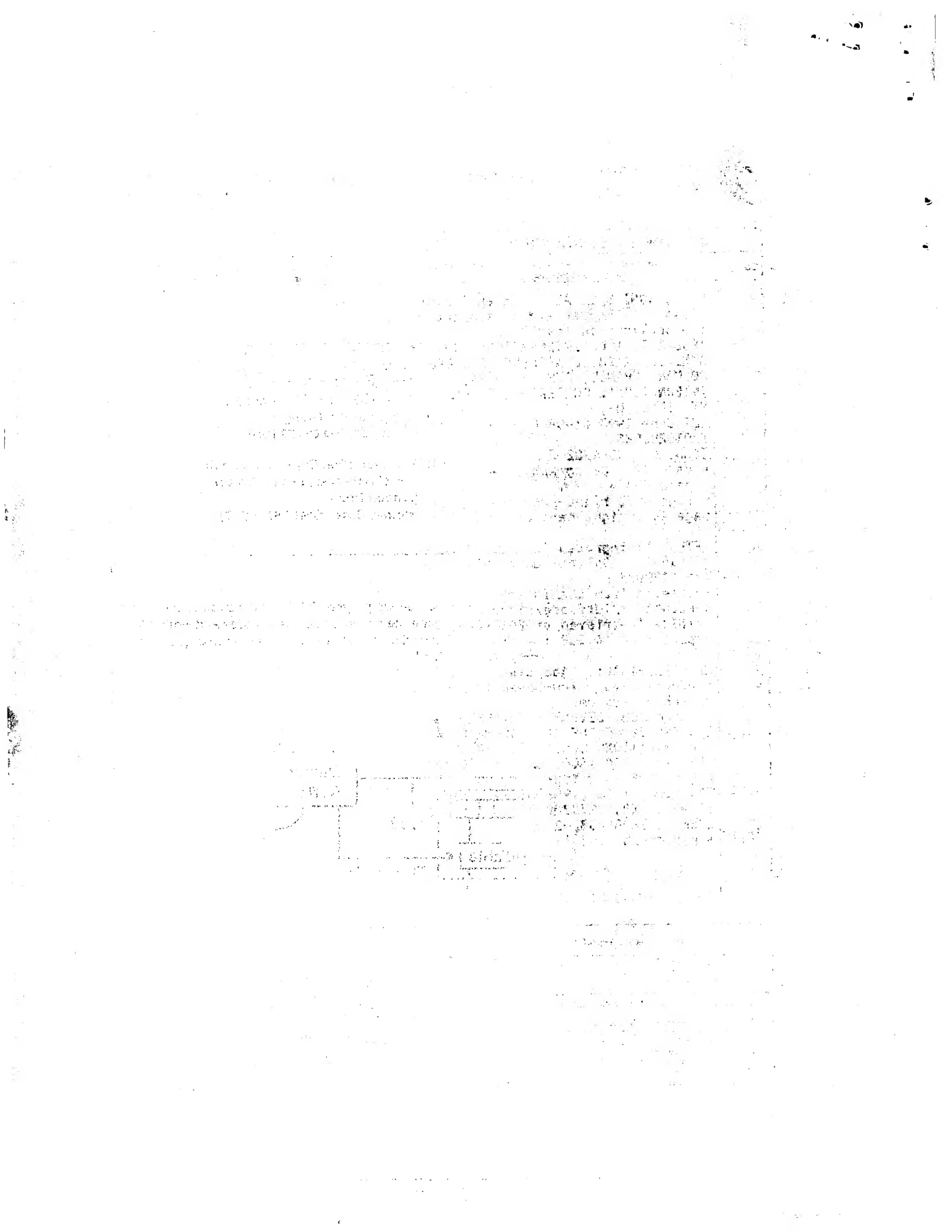
CATEGORY OF CITED DOCUMENTS

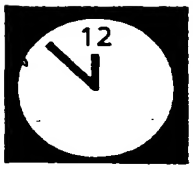
X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)





Web Server Support for Tiered Services

Nina Bhatti and Rich Friedrich
Hewlett-Packard Research Labs

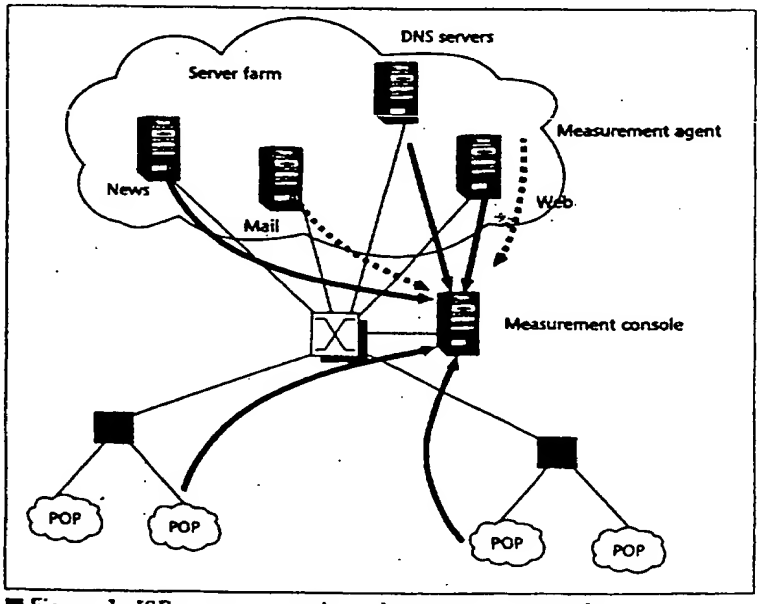
Abstract

The evolving needs of conducting commerce using the Internet requires more than just network quality of service mechanisms for differentiated services. Empirical evidence suggests that overloaded servers can have significant impact on user perceived response times. Furthermore, FIFO scheduling done by servers can eliminate any QoS improvements made by network-differentiated services. Consequently, server QoS is a key component in delivering end to end predictable, stable, and tiered services to end users. This article describes our research and results for WebQoS, an architecture for supporting server QoS. We demonstrate that through classification, admission control, and scheduling, we can support distinct performance levels for different classes of users and maintain predictable performance even when the server is subjected to a client request rate that is several times greater than the server's maximum processing rate.

Much of the current research and engineering in the field of quality of service (QoS) has focused on network performance attributes [1]. This has been principally motivated by the application requirements for isochronous streams which have strict constraints on bandwidth, delay, and jitter. The Internet Engineering Task Force (IETF) [2, 3] has codified proposals for network QoS with the goal of supporting differentiated services.

As Internet usage grows it has become apparent that non-isochronous applications such as delivering static and dynamically generated Web pages can also benefit from QoS. As demonstrated in North America during the holiday season in late 1998, the market for electronic commerce increased dramatically. This unexpected and unpredictable increase in client demand resulted in increased congestion and queuing in the servers. Consequently, server response times grew and consumer patience shrunk. Additionally, some applications desire preferential treatment for users or services which is foreign to the egalitarian philosophy of TCP/IP and HTTP. For example, during the U.S. stock market panic of October 1998 large and small investors alike were greeted with "server busy" errors from online trading companies. For these applications significant revenue was lost by not being able to process large trades. As these examples illustrate, network QoS by itself is not sufficient to support end to end QoS. The servers must also have mechanisms and policies for establishing and supporting QoS.

We hypothesized that servers would become an important element in delivering QoS. In 1997 our research set out to answer the following questions related to making Internet-based applications predictable and stable so that they could support the impending electronic economy. These Internet applications have a potential client population in the millions of users and it is important to understand what requirements they place on servers and networks.



■ Figure 1. *ISP server, network, and measurement topology.*

0890-8044/99/$10.00 © 1999 IEEE

- What is the impact of Internet workloads on servers?
- What is the impact of server latency on end to end latency?
- What server mechanisms are necessary to improve QoS?
- How can servers be protected from overload?
- How can servers support tiered (differentiated) user service levels with unique performance attributes?

In this article we will examine these questions and provide results based on empirical observations of our WebQoS research prototype. We will show that the servers play an increasing role in providing end to end QoS and that tiered services can enable new application capabilities for existing Internet-based applications such as the Web and e-commerce.

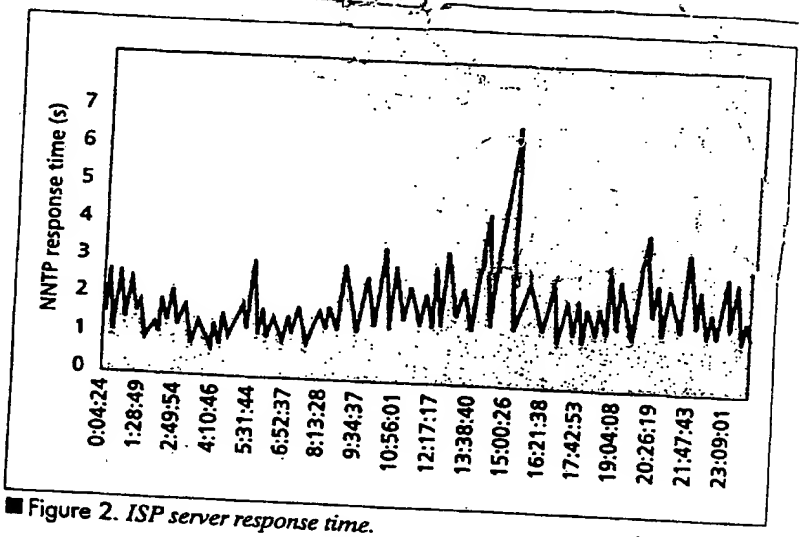
Servers and QoS

In the summer of 1997 HP Labs researchers had the opportunity to instrument and monitor one of the largest ISPs in North America. This research allowed us to quantify the delay components for Web, news, and mail services. A simplified topology of the ISP's network and data center is shown in Fig. 1.

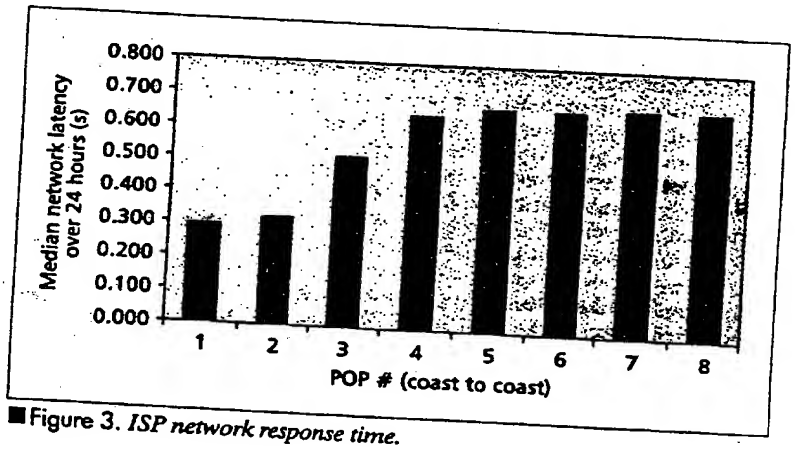
The data center contained hundreds of servers supporting Web, news, mail, and other IP-based services. Clients accessed these systems through 200 points of presence (POPs) located in the United States.

A mixture of active and passive measurement techniques were used to collect performance metrics [4]. The active techniques simulated actual users by periodically issuing predefined requests. For example, the news servers were tested with a client-based script that always returned a known 40-kbyte response. The passive measurements derived data from performance counters and log files in applications and system software.

A representative sample of measurements for this well engineered network are shown in Figs. 2 and 3. In Fig. 2 the Network News Transfer Protocol (NNTP) server response time for requests of 40-kbyte-sized articles are plotted during a 24-hr period. Most of the time the smoothed response time was on the order of 1 to 2 s with a peak near 7s. Figure 3 plots the coast-to-coast network response for a 40-kbyte transfer for eight of the busiest POPs. The median value ranges from 300 ms to 700 ms. From the data collected at this large ISP site we concluded that servers were currently a significant component in end-to-end delay.



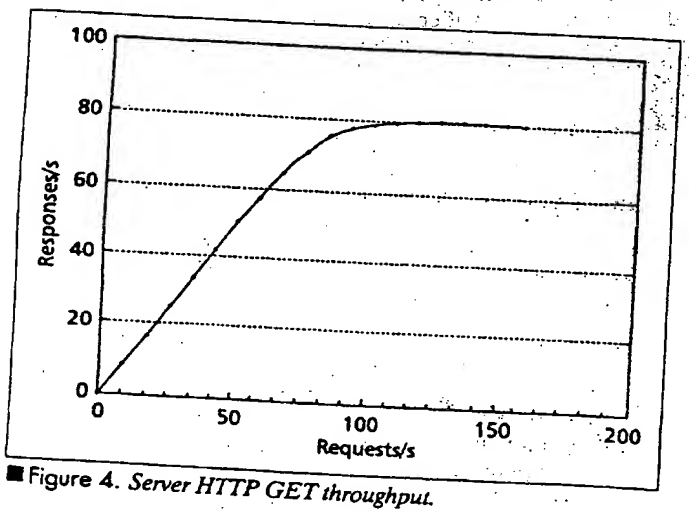
■ Figure 2. *ISP server response time.*



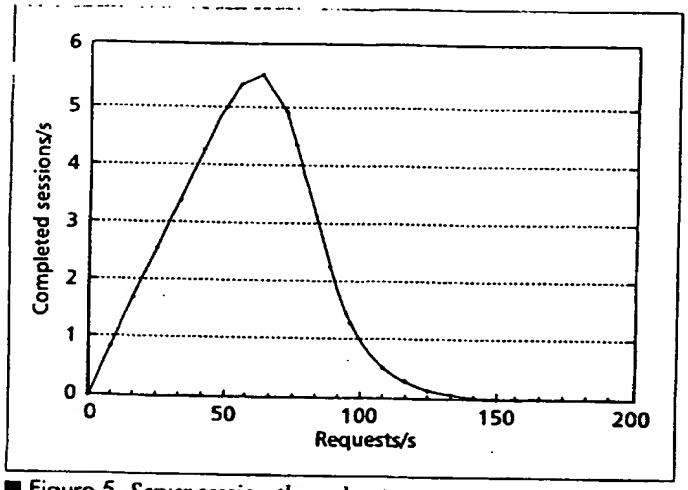
■ Figure 3. *ISP network response time.*

E-commerce applications are more complex than this simple news workload. They consist of client, network, Web server, and application components, each of which contributes to the end to end performance. Several trends are affecting network and server performance for these complex applications. First, network latency is decreasing as the backbone moves toward 622 Mb/s and 1 Gb/s bandwidth. Second, some ISPs provide guaranteed network latency, and in the future private peering agreements between backbone providers will further improve the performance of IP traffic. As caches become more pervasive, static content response time will be improved. Furthermore, client access network latency is improving two to 10 times with the rapid adoption of integrated services digital network (ISDN), digital subscriber line (DSL), and cable modems.

At the same time, there are several trends that are increasing server latency time for sophisticated Internet applications. First, flash crowds can overload a popular server, leading to sluggish response times or, worse yet, an inability for clients to connect at all [5]. Furthermore, servers can become so busy that they end up wasting resources by processing requests only to find too late that the user is no longer interested and has terminated the TCP/IP connection. Second, new application technologies also increase the processing demands on the



■ Figure 4. *Server HTTP GET throughput.*



■ Figure 5. *Server session throughput.*

server. Some of these new technologies include Java, Secure Socket Layer (SSL) for security, dynamic data, database transactions, and sophisticated application middleware components. Third, the media has also become much richer with larger and more images being used as sites try to woo consumers by distinguishing their site from others. Audio, voice, and video are also becoming more prevalent, and it is estimated that they will become a significant portion of all IP traffic in the next few years.

We also discovered that overloaded servers are a major force in poor end-to-end QoS. Our measurements of busy Web sites found the following problem. In Fig. 4 the server response rate versus the HTTP GET rate is plotted for a typical Web server. As expected, when measured from an HTTP request perspective, the response rate grows linearly until the server nears maximum capacity. At this point an asymptote is reached, and the maximum throughput of the server is unchanged over a range of increasing request rates. In Fig. 5 the same data is presented but now analyzed from a client's perspective; it is apparent that session throughput collapses when servers become overloaded. This *session view* represents the sequence of HTTP requests that occur between a client browser and a Web server in order to download all required pages and logically complete a user transaction. For example, a session might include browsing for a book title, adding items to a shopping cart, and paying with a credit card. From this session perspective the session throughput collapses rapidly as the server becomes busy due to queuing and congestion on the server. This queuing results in timeouts and clients who give up and abort the request. Furthermore, longer-length sessions, which have more commercial value since they are more likely to result in a financial transaction, have a dramatically lower throughput compared to shorter sessions since the probability that an HTTP request in this sequence is lost grows exponentially as the session length increases. Note that the network performance is not a factor in this scenario.

But why not overprovision the server and not have to deal with the issue? Unfortunately, if one examines the evolution of Web applications, one finds a steep growth in the client demand curve that makes provisioning problematic and not conducive to static resource allocation approaches. Historically, applications executing on a mainframe supported a few thousand clients. These applications evolved to a client–server world of tens of thousands of clients. Now Internet application environments are faced with a client population that appears essentially infinite. Consequently, brute force server resource provisioning is not fiscally prudent since no reasonable amount of hardware can guarantee predictable performance for flash crowds. Furthermore, overprovisioning of servers cannot provide tiered services for users or applications that require preferential treatment.

Furthermore, while network QoS provided by differentiated services is a critical component for the predictability and stability of an Internet-based application, it is only part of the picture. Intelligent network bandwidth management and congestion avoidance features cannot resolve scheduling or bottleneck problems at the server. Networking devices that perform classification operations and bandwidth reservation to speed along packets are undermined by the prevalent first-in-first-out (FIFO) scheduling policies used in most UNIX kernels since a busy server indiscriminately drops high priority network packets [6]. Network QoS and its associated packet priorities are ineffective when the server drops such a packet. Supporting tiered services on a user or application basis requires server QoS mechanisms. Clearly, the server must play an increasingly important role in delivering end-to-end QoS to both provide overload protection and to also enable tiered service support.

Architecture for Server-Based QoS

Our WebQoS server architecture has focused on an application environment that consists of multiple nodes supporting Web servers, application servers, and database servers. Our philosophy was to create a low overhead, scalable infrastructure that was transparent to applications and Web servers.

Our goal was to support two key capabilities. First, the architecture must effectively manage peaks in client HTTP request rates. Second, it must support tiered service levels that enable preferential treatment of users or services. In order to support tiered service levels we observed that without classes, best-effort traffic competes equally with premium traffic for scarce resources. This results in poor performance for all users. Our solution uses admission and scheduling control to improve performance for premium tiers. The architecture is shown in Fig. 6. This architecture accepts incoming requests, reorders them based on scheduling policies, and transparently submits them to the Web server for normal processing.

In order to process requests according to their importance the request's class must be determined. The *request classification* sets QoS attributes based on the filters and policies defined by the system administrator. Requests from *premium* users are classified as high priority and given preferential processing treatment. After a request is classified the request can be rejected or executed according to the scheduling policies appropriate to each class. The *admission control* component determines if this request should be processed. If so, the classification attributes are used by *request scheduling* to determine how to enqueue and dispatch this request. *Session management* is used to provide session semantics and maintain session state for the stateless HTTP protocol. Additional mechanisms can be used to control resource allocation on the server. *Resource scheduling* ensures that high-priority tasks are allocated and executed as high-priority processes by the host operating system.

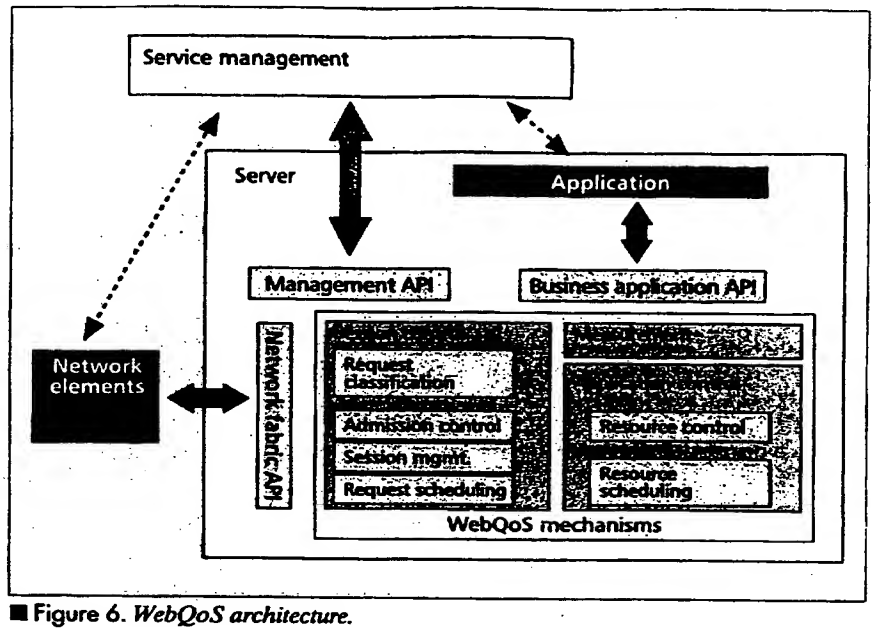
The architecture also supports integration with network QoS mechanisms such as reading and marking IP type of service (ToS) or differentiated services fields which can give drop preference behavior for TCP connections. We also support integration with management systems so that WebQoS configuration parameters can be remotely set and internal measurements can be exported for monitoring purposes.

We realized this architecture in a prototype that intercepts, classifies, queues, and schedules the TCP/IP requests transparently to an Apache Web server.

Related Work

Other efforts have been made to implement differentiated services on Web servers. In [7], *resource containers* are used to account for and control consumption of operating system resources by different classes of requests [7]. This operating system control mechanism has been shown to ensure class-based performance in Web servers. Another approach is the scheduling of Web server worker processes with the same priority as the request [8]. This ensures that the higher-priority requests are executed first once they are assigned to a worker process.

External devices such as intelligent switches or routers can be placed in front of a Web server to intercept Web traffic. Alteon, Packeteer, and Cisco Local Director switch products can be configured to perform request-aware traffic shaping. Typically, these devices are used as load balancers for a cluster of Web servers. Other switches can differentiate performance based on application type using admission control and bandwidth reservation [9].



■ Figure 6. *WebQoS architecture.*

Prototype

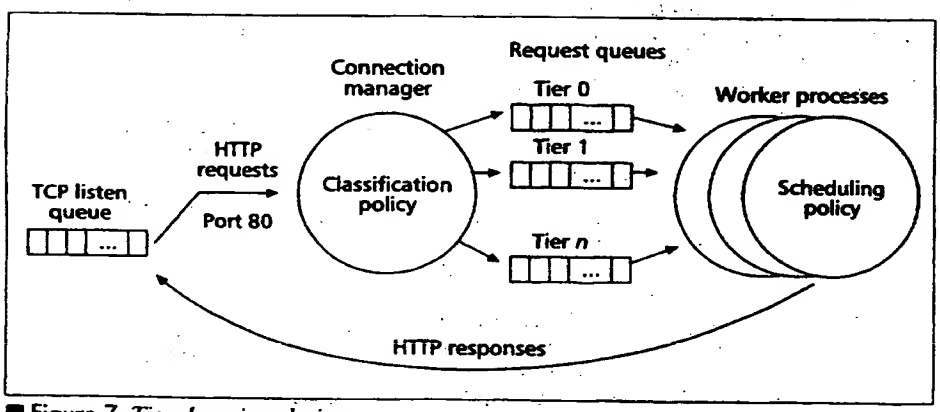
To achieve preferential treatment of different classes we modified the FIFO servicing model of a popular Web server, Apache 1.2.4 [10]. This typical Web server is composed of a collection of identical worker processes that listen on a UNIX socket for HTTP connections and serve requests as they are received.

In this section we discuss the major components of the prototype. These include a connection manager, request classifier, admission controller, request scheduler, and resource scheduler.

Connection Manager

We modified Apache by creating a new unique acceptor process, the *connection manager*, that intercepts all requests. This process classifies the request and places the request on the appropriate tier queue. The Apache worker processes receive requests from these queues instead of directly from the HTTP socket. A worker process selects the next request based on the scheduling policy. For example, work from the top tier will be processed first for priority scheduling. This bit of indirection allows our scheduling policies to decide which requests are processed first and which requests can be rejected. Figure 7 illustrates the design of the prototype.

Since all requests must be accepted by the connection manager it is essential that the connection manager runs frequently enough to keep request queues full. If the connection manager does not run frequently enough worker processes may execute requests from lower tiers because all the requests from higher tiers have been processed even though there may be top-tier requests waiting to be accepted. This results in a server that is more "fair" but may allocate server processes to lower-priority work, and thus make it difficult to quickly respond to newly arriving premium requests, thereby violating the preferential treatment policy. In addition, if the connection manager does not run frequently enough requests may fill the TCP listen queue until its limit is reached and no other TCP connections will be allowed to the HTTP port. Consequently, premium requests are prohibited from establishing a TCP connection, and are thus dropped.

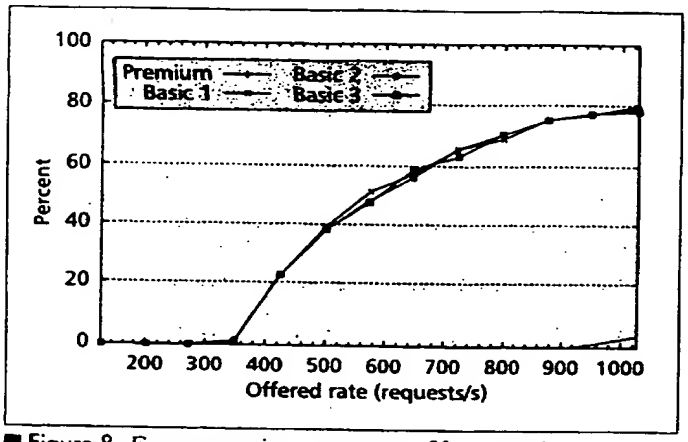


■ Figure 7. *Tiered services design.*

Request Classification

A key requirement to support tiered services is the ability to identify and classify the incoming requests of each class. There are several ways to classify requests. These classification mechanisms can be divided into two categories, user class-based or target class-based. User class-based classification characterizes requests by the source of the request; target class-based classification classifies by the content or destination of the request. Specifically, we support the following classification filters.

- User class-based classification:
  - The client IP address is used to distinguish one individual client from another. This method is the simplest to implement. However, the client IP address can be masked due to proxies or fire walls, so this method has limited application.



■ Figure 8. *Errors, premium constant at 50 requests/s.*

- HTTP cookies, a unique identifier sent to the browser, can be embedded in the request to indicate to which class the client belongs. For example, a subscription to a particular service is implemented as a persistent cookie. A cookie can also be used to identify a session that has been established for session-based classification.
- Browser plug-ins can also embed special client identifiers in the body of each HTTP request. Such a plug-in could be downloaded by clients who have paid for a subscription to premium service.

- Target class-based classification:
  - The URL request type or filename path can be used to classify the relative importance of the request. In this case the sender of the request is irrelevant. Content can be classified as mission-critical, delay-sensitive, or best-effort. This would allow e-commerce purchase activities, for example, to have higher priority than browsing activities.
  - Destination IP addresses can be used by a server when the server supports co-hosting of multiple destinations (Web sites) on the same node.

Admission Control

When the server processing rate falls behind the client demand rate, the server becomes unresponsive to both premium and basic classes. To protect the server from high client loads some requests must be rejected. Naturally, basic requests rather than premium requests should be rejected first, and existing sessions should be maintained. Admission control of basic requests is triggered when the server starts to be loaded.

We have two admission control trigger parameters, the first based on the total capacity of the server and the second based only on its ability to keep up with the performance demand of premium tiers. During our experiments we found that when too many total requests entered the server premium tier, performance suffered so lower tier requests are rejected to shed load. The second triggers based on the number of premium requests waiting, since premium tier responsiveness is essential.

- Total requests queued: When the WebQoS environment reaches a programmable trigger point, j, the server will reject basic requests until the total number of queued requests falls below j. Note that this limit is based on all requests queued, not on requests for any particular class.
- Number of premium requests queued: When the WebQoS environment reaches a programmable trigger point, k, the server will reject all basic requests until the number of queued premium requests falls below k. This rejection policy is sensitive to the waiting time of premium requests. As long as premium requests are not waiting, basic requests can be accepted.

Rejection in the prototype is done by simply closing the connection, which results in a "connection reset by peer error." A more client-friendly server could instead return a customized "sorry, server is too busy, please try later" HTML response (although this will consume additional CPU and network resources that might be scarce during periods of overload).

Request Scheduling

After requests are classified according to one of the above classification schemes and admitted by admission control, the server must actually realize different service levels for each class of requests. This is done by selecting the order of request execution. Workers are autonomous processes that select requests to process based on the scheduling policy. The scheduling policy may depend on queue lengths. Worker processes may be able to execute requests from any class, or, to reserve capacity for higher-class processes, they may be restricted to executing premium-class traffic. Below we outline several potential policies. To preserve Apache worker process design, after a request has been selected it will run until completion. Only then is the worker available to process another request.

- *Strict priority* schedules all higher-class requests before lower-class requests even when low-priority requests are waiting.
- *Weighted priority* schedules a class based on its weighted importance. For example, one class will get twice as many requests scheduled if its class weight is twice another's.
- *Shared capacity* schedules each class to a set capacity, and any unused capacity can be given to another class. The class may also have a minimum reserve capacity that cannot be assigned to another class.
- *Fixed capacity* schedules each class to a fixed capacity that cannot be shared with another class.
- *Earliest deadline first* establishes schedules based on the deadline for completion of each request. This can be used to give predicted response time guarantees.

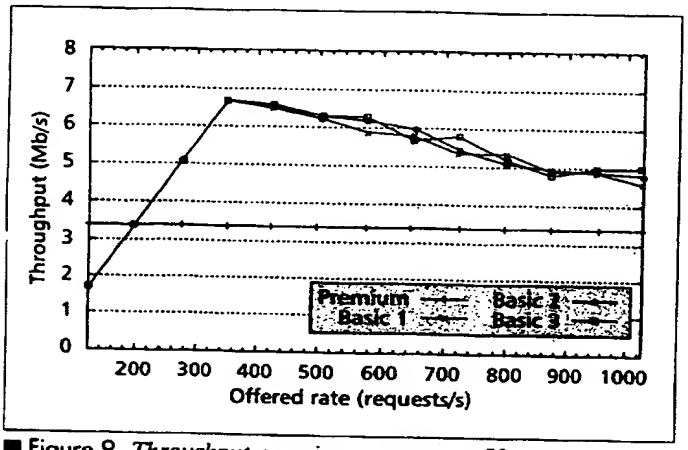
Resource Scheduling

To provide even more control over the execution of requests, several methods can be used to favor the execution of workers executing premium requests and retard the execution of basic requests. The server can set priorities on worker threads or processes to match the priorities of the request. The "nice" UNIX system call can give more CPU to higher tiers. Finally, the HP-UX kernel facility Process Resource Manager, which is built upon a fair share scheduler, can control each worker's share of system resources. Each of these schemes can be either statically set up when a worker process is created or dynamically changed depending on the request class being executed.

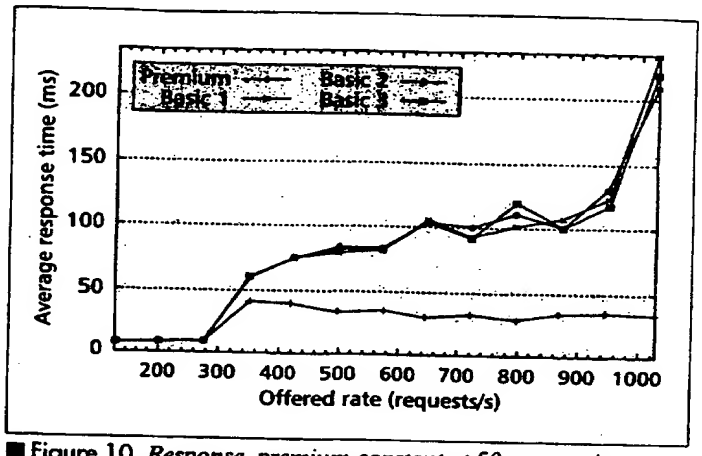
Apache Source Modifications

In keeping with our philosophy of application transparency the number of Apache code changes is minimal. The `http_main.c` file has small modifications to start the connection manager process, set up queues (socket pairs), and change the child Apache process to accept requests from the connection manger process instead of directly from the HTTP socket. One additional file containing about 900 lines of C, `connection_mgr.c`, is linked when building the Apache server. This new file contains the classification policy, enqueue mechanisms, dequeue policies, and connection manager process code.

Additional shared memory and semaphores are required for our prototype. Shared memory is used for the state of the



■ Figure 9. *Throughput, premium constant at 50 requests/s.*



■ Figure 10. *Response, premium constant at 50 requests/s.*

queues and contains each class queue length, number of requests executing in each class, last class to have a request dequeued, and the total count of waiting requests on all classes. Access to shared memory is synchronized through the use of a semaphore, and one additional semaphore is used to signal waiting workers when requests are available.

Results

In this section we evaluate the performance of the Apache-based prototype. We give comparisons of response time, throughput, and error rates for premium and basic clients running with priority scheduling. We compare the performance of premium and basic clients where the premium request rate is fixed and also with the premium request rate identical to the basic clients. We will demonstrate in both cases that premium clients receive much better service quality than basic clients. To further illustrate the point we compare a more real world metric, session throughput, of premium clients with basic clients.

Test results are based on four client machines simulating many clients issuing uniform size HTTP GETs. The experimental setup is as follows:

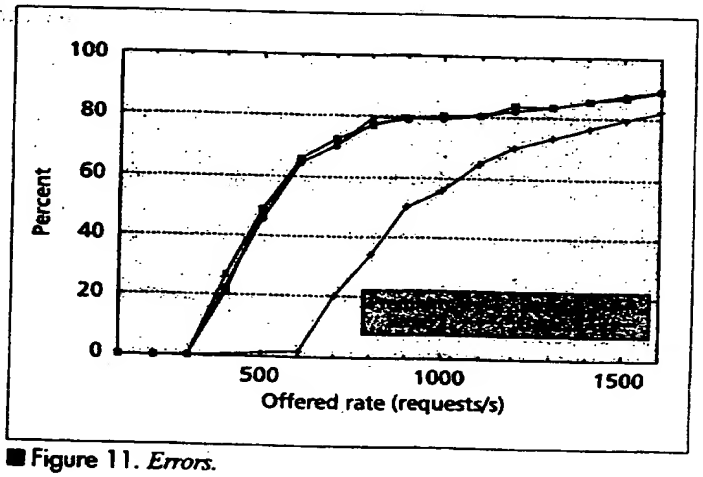
- **Server:** Single-processor HP 9000 K460 (PA-8200 CPU) running HP-UX 10.20, 512 Mbytes main memory, 100-BaseT network connection. The size of the listen queue is set to 32.[1] Apache was configured to run with 32 worker processes.
- **Clients:** Four HP 9000 workstations (two 735/99 and two 755/99) running HP-UX 10.20, 100-BaseT network connection. `httperf` is used on each client to generate the workload [11].
- **Network:** 100-BaseT Ethernet through an HP AdvanceStack Switch 2000.

The four clients communicate with the server over the 100-BaseT Ethernet. On each client, the `httperf` application is configured to issue requests at a given fixed rate (e.g., 50 requests/s) and for a given static Web page of 8 kbytes for a 5-min period. Each request has a five second timeout, and if no response arrives from the server within that time period, the client aborts the connection and logs an error. This effectively sets an upper limit (excluding connection setup time) on response delay as measured at the client. This will be evident in the graphs. At the end of the 5-min period, statistics are collected at each client (number of successfully completed requests, average response time for these requests, number of requests that did not complete successfully) and at the server (CPU utilization). The client request rates are then increased and the process is repeated, yielding a graph that shows performance as a function of increasing offered load.

Our results are from the Apache prototype using premium tier strict priority scheduling, that is, lower tiers are only executed if no work exists in any higher tiers. We also configured admission control to trigger rejection when there are j or more total number of queued requests, or when there are k or more premium requests waiting. We evaluated the effect of these parameters with file requests for a variety of sizes and rates. Through experimentation we determined optimal j and k values for our server test configuration.

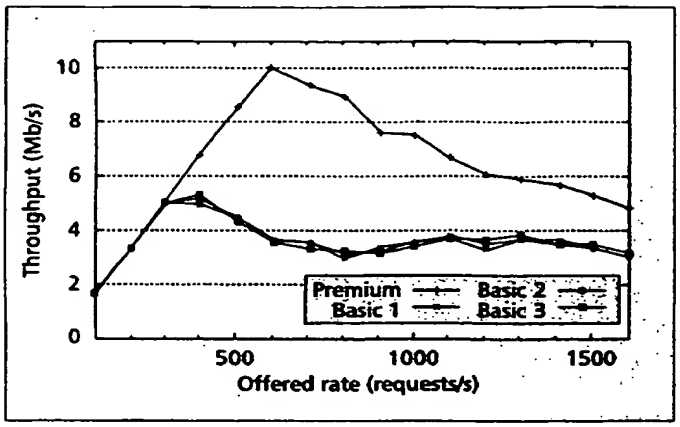
Three graphs illustrate experimental results for each test case. The first, Fig. 8, plots the percent of client requests that encounter an error as a function of the total offered client demand rate (which is equal to the sum of premium and basic clients). The second graph, Fig. 9, plots the data throughput in Mb/s as a function of the total offered client demand rate. Finally, the third graph, Fig. 10, plots the average response time in ms for all completed client requests as a function of the total offered client demand rate.

In the first set of tests we analyze the errors, byte throughput, and response time of a server handling one premium client at a fixed rate of 50 requests/s and three basic clients with request rates that monotonically increase from 25-325 requests/s. This results in a maximum combined client demand rate of 1025 requests/s. All client requests are for a

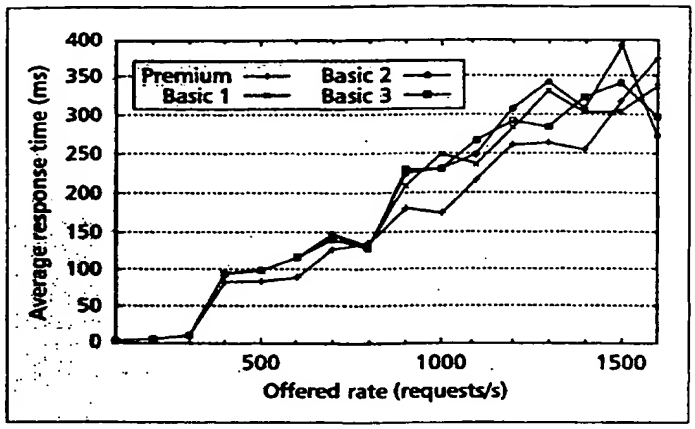


■ Figure 11. *Errors.*

[1] *The listen queue is set with the listen() system call, and when set with a value the actual queue length is 1.5 times that value. In this case we are setting a value of 48.*



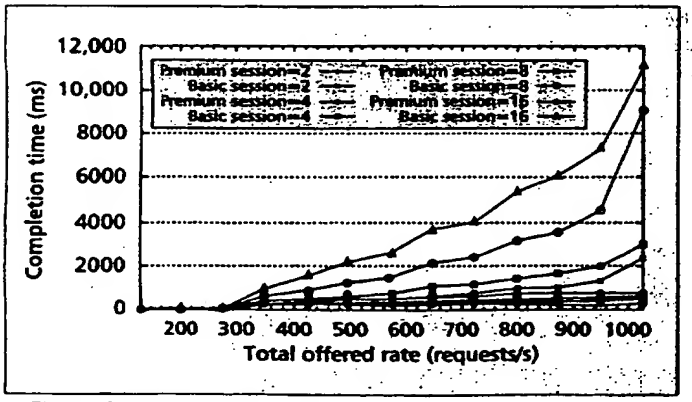
■ Figure 12. *Throughput.*



■ Figure 13. *Response.*

fixed object size of 8 kbytes. This experiment models a situation where there are a fixed number of clients with a subscription for premium service. We hoped to verify if a premium load can be protected while more and more basic traffic is added. As shown in Fig. 8, at low request rates the server can easily satisfy all client requests and premium and basic clients receive excellent service. As the offered load increases the basic clients consume more and more bandwidth until the server's capacity is exhausted at around 350 requests/s. At this point the basic client requests start to be rejected to shorten the premium queue length and thus waiting time. Notice that the errors start to climb only for requests from basic service tiers, as shown in Fig. 8. The premium client does not experience any errors until the very end of the test at 1025 requests/s (which is nearly three times the maximum capacity of the server). At this point the premium client service experiences a small error rate of only 3 percent (note that the premium error rate may be difficult to read since it is at zero for most of this test). The premium throughput is very well protected as it appears as a straight line at 3.4 Mb/s, while the basic request throughput reaches a maximum at 350 requests/s and then continues to drop despite an increasing request rate, as shown in Fig. 9. In Fig. 10 the average response time for premium clients is also protected while the basic response time starts to climb sharply above 300 requests/s.

The second performance measurements are for premium and basic clients that monotonically increase with each client issuing an equal number of requests. These results are shown in Figs. 11, 12, and 13. All four clients have the same request rates for a given point on each of these graphs. Thus, the proportion of basic to premium requests is 3:1.



■ Figure 14. *Session completion time.*

For this experiment, resource allocation at the server is given preferentially to premium users, and they always have a larger fraction even when the load is well beyond the maximum capacity of the server, as shown in Fig. 12. The premium clients have better throughput and somewhat better error rates. The response time is somewhat misleading since basic requests are only processed when the server can process them quickly; otherwise basic requests are rejected and the response time is not counted. Thus, the response time plotted here is optimistic since it does not include requests that are rejected or encounter an error. The error rates are not as different as we would like. All the errors for premium clients are due to timeouts that we set to 5 s in the client load generator.

Finally, the session completion times are shown in Fig. 14. This analysis builds on the above results to evaluate a more complex interaction, such as a page download or credit card payment. For this graph the session length is defined to be a sequence of N consecutive HTTP requests with minimal interarrival time. N ranges from 2 to 16. Completion time (i.e., total session duration) is plotted in milliseconds as a function of the total client request rate. The premium session completion times are the lower four lines; the steeply climbing top four lines are the session completion times of basic client sessions. For a session length of 2 the premium and basic response times start to diverge above 350 requests/s. At a request rate of 650 the premium session completion time is 60 ms, while the basic session completion time has grown to 530 ms. Similarly, for a session length of 16 the premium and basic response times start to diverge above 350 requests/s. At a request rate of 650 the premium session completion time is 450 ms, while the basic session completion time has grown to 3550 ms. With more complex Web pages the differentiation between premium and basic session completion times will grow. Workload measurements that we have conducted of major Web sites indicate that a single page download may require as many as 50 HTTP requests, so the service quality differentiation results shown in Fig. 14 are less than what a typical user is likely to encounter on the World Wide Web.

Summary

In this article we highlight the need for server QoS mechanisms to support tiered user service levels and protect servers from client demand overload. We develop an architecture, WebQoS, that can support these QoS benefits transparently to a TCP/IP-based application. Finally, we discuss a prototype that realized the architecture and illustrated its benefits through experimental results. Specifically, we demonstrate

that it can provide preferential treatment to premium users for throughput, response time, and error rates.

There are still many unsolved problems in providing tiered service levels end to end. The first is a tighter integration of server and network QoS and the ability to communicate QoS attributes across network domains. The applicability of QoS routing to this problem also needs evaluation. Second, more flexible admission control mechanisms might employ a continuum of content adaptation instead of request rejection. This would result in the benefit of a server returning degraded content instead of rejecting a request [12]. Third, lightweight signaling mechanisms are needed to ensure that high priority traffic is given preferential treatment along the path between client and server. This is especially important for e-commerce workloads where TCP session times are very short. Finally, the implications of end-to-end QoS on devices at the edge of the network should be explored to determine which benefits can be derived from them.

Acknowledgments

The authors would like to thank the WebQoS Team — Martin Arlitt, John Dilley, Tai Jin, David Mosberger, Jim Salehi and Anna Zara — for their efforts in refining the notion of tiered Web services and for creating the experimental testbed we used to evaluate our ideas. We are also indebted to Ed Perry, Srinivas Ramanathan, and Preeti Bhoj for their ISP measurement techniques and results.

References

[1] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A Survey of QoS Architectures," ACM/Springer Verlag *Multimedia Sys. J*, Special Issue on QoS Architecture, vol. 6, no. 3, May 1998, pp. 138–51.
[2] An Architecture for Differentiated Services, IETF, Oct. 1998.
[3] Y. Bernet *et al.*, "A Framework for End-to-End QoS Combining RSVP/Intserv and Differentiated Services," IETF, Mar. 1998.
[4] C. Darst and S. Ramanathan, "Measurement and Management of Internet Services," *6th IFIP/IEEE Int'l. Symp. Integrated Network Mgmt.*, Boston, MA, May 1999, pp. 125–40.
[5] J. Mogul and K. K. Ramakrishnan, "Eliminating Receive Likelock in an Interrupt-Driven Kernel," *ACM Trans. Comp. Sys.*, Aug. 1997, pp. 217–252.
[6] P. Druschel and G. Banga, "Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems," *Proc. 2nd USENIX Symp. Op. Sys. Design and Implementation*, Oct. 1997.
[7] G. Banga and P. Druschel, "Resource Containers: A New Facility for Resource Management in Server Systems," *Proc. 3rd USENIX Symp. Op. Sys. Design and Implementation*, New Orleans, LA, Feb. 1999.
[8] J. Almeida *et al.*, "Providing Different Levels of Service in Web Content Hosting," *Proc. the Internet Server Perf. Wksp.*, Mar. 1998.
[9] V. Srinivasan *et al.*, "Fast and Scalable Layer Four Switching," *Proc. SIGCOMM '98 Symp.*, Vancouver, Canada, Sept. 1998.
[10] Apache Group, http://www.apache.org
[11] D. Mosberger and T. Jin, "httperf: A Tool for Measuring Web Server Performance," *Proc. Internet Server Perf. Wksp.*, Madison, WI, June 1998, pp. 59–67.
[12] T. Abdelzaher and N. Bhatti, "Web Server QoS Management by Adaptive Content Delivery," *7th Int'l. Wksp. QoS*, May 1999.

Biographies

NINA BHATTI [M] (nina@hpl.hp.com) is a researcher at Hewlett-Packard's Internet Systems and Applications Laboratory in Palo Alto, California. Her research interests are Internet technologies, quality of service, distributed systems, and network protocols. She received a B.A. in computer science from the University of California, Berkeley, and a Ph.D. and M.S. from the University of Arizona. She is a member of ACM.

RICH FRIEDRICH (richf@hpl.hp.com) is a senior researcher at Hewlett-Packard Laboratories. During his 17 years at HP he has led projects that developed and optimized the first commercial RISC-based systems, the first multi-processor OLTP RISC systems, and a distributed measurement system for the OSF Distributed Computing Environment. He is currently focusing on server control QoS mechanisms and for Internet e-commerce services. In 1998, he was co-chair of the International Workshop of Quality of Service. He is a member of the IEEE.